

UNIVERSITY OF EDINBURGH
COLLEGE OF SCIENCE AND ENGINEERING
SCHOOL OF INFORMATICS

INFR08013 INFORMATICS 1 - FUNCTIONAL PROGRAMMING

Monday 0 August 2017

09:30 to 11:30

INSTRUCTIONS TO CANDIDATES

1. Note that **ALL QUESTIONS ARE COMPULSORY**.
2. **DIFFERENT QUESTIONS MAY HAVE DIFFERENT NUMBERS OF TOTAL MARKS**. Take note of this in allocating time to questions.
3. This is an **OPEN BOOK** examination: notes and printed material are allowed, and **USB sticks (read only)**, but no electronic devices.
4. **CALCULATORS MAY NOT BE USED IN THIS EXAMINATION**

Convener: I. Simpson
External Examiner: I. Gent

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

1. (a) Write a function `f :: String -> [Int] -> String` that repeats each character in its first argument the number of times given by the number at the corresponding position in its second argument, with negative numbers counting the same as 0. If the lengths of the two lists do not match, the extra elements in the longer list are ignored. For example:

```
f "abcde" [3,1,2,0,4] = "aaabcceeee"
f "call" [3,-2,1,2,7] = "cccllll"
f "raisin" [1,2,3,4] = "raaiiissss"
f "moose" [2] = "mm"
f "" [1,2,3] = ""
```

Use *basic functions*, *list comprehension*, and *library functions*, but *not recursion*. Credit may be given for indicating how you have tested your function.

[16 marks]

- (b) Write a second function `g :: String -> [Int] -> String` that behaves like `f`, this time using *basic functions* and *recursion*, but *not list comprehension* or *library functions*. Credit may be given for indicating how you have tested your function.

[16 marks]

2. (a) Write a function `p :: String -> Bool` that checks whether or not all the digits appearing in a string are odd. For example:

```
p "Inf1-FP"      = True
p "Functional"   = True
p "1+1=2"        = False
p "3.157/3 > 19" = True
```

Use *basic functions*, *list comprehension*, and *library functions*, but *not recursion*. Credit may be given for indicating how you have tested your function.

[12 marks]

- (b) Write a second function `q :: String -> Bool` that behaves like `p`, this time using *basic functions* and *recursion*, but *not list comprehension* or *library functions*. Credit may be given for indicating how you have tested your function.

[12 marks]

- (c) Write a third function `r :: String -> Bool` that also behaves like `p`, this time using one or more of the following higher-order library functions:

```
map      :: (a -> b) -> [a] -> [b]
filter  :: (a -> Bool) -> [a] -> [a]
foldr   :: (a -> b -> b) -> b -> [a] -> b
```

You may use basic functions but do *not* use *recursion*, *list comprehension* or *library functions* other than these three. Credit may be given for indicating how you have tested your function.

[12 marks]

3. This question concerns simple commands for moving a robot back and forth along an infinite line like this:

. . . $\frac{-3}{|}$ $\frac{-2}{|}$ $\frac{-1}{|}$ $\frac{0}{|}$ $\frac{1}{|}$ $\frac{2}{|}$ $\frac{3}{|}$. . .

The following declarations define the commands:

```
data Move =
  Go Int          -- move the given distance in the current direction
  | Turn         -- reverse direction
  | Dance        -- dance in place, without changing direction

data Command =
  Nil            -- do nothing
  | Command :#: Move -- do a command followed by a move
```

Before and after each move, the robot is in some “state” — in a position on the line, and facing left or right:

```
type Position = Int
data Direction = L | R
type State = (Position, Direction)
```

The above declarations are provided in the template file, together with code to print values of type `Move`, `Command` and `State` and to compare them for equality, along with code that enables QuickCheck to generate arbitrary values of type `Move`, `Command` and `State`, to aid testing.

- (a) Write a function `state :: Move -> State -> State` that, given a move and the current state of the robot, returns the state of the robot following the move. For example:

```
state (Go 3) (0,R) = (3,R)
state (Go 3) (0,L) = (-3,L)
state Turn (-2,L) = (-2,R)
state Dance (4,R) = (4,R)
```

Credit may be given for indicating how you have tested your function. [8 marks]

QUESTION CONTINUES ON NEXT PAGE

QUESTION CONTINUED FROM PREVIOUS PAGE

- (b) Write a function `finalstate :: Command -> State -> State` that, given a command and the current state of the robot, returns the state of the robot following the command. For example:

```
finalstate (Nil) (3,R) = (3,R)
finalstate (Nil :#: Go 3 :#: Turn :#: Go 4) (0,L) = (1,R)
finalstate (Nil :#: Go 3 :#: Turn :#: Dance :#: Turn) (0,R) = (3,R)
finalstate (Nil :#: Go 3 :#: Turn :#: Go 2 :#: Go 1 :#: Turn :#: Go 4) (4,L)
= (0,L)
```

Credit may be given for indicating how you have tested your function. *[8 marks]*

- (c) Write a function `simplify :: Command -> Command` that converts a command to an equivalent command by combining successive `Go` moves and combining successive `Turn` moves. For example:

```
simplify Nil = Nil
simplify (Nil :#: Go 3 :#: Go 1 :#: Go 4 :#: Dance)
= Nil :#: Go 8 :#: Dance
simplify (Nil :#: Go 3 :#: Turn :#: Dance :#: Turn)
= Nil :#: Go 3 :#: Turn :#: Dance :#: Turn
simplify (Nil :#: Go 3 :#: Turn :#: Turn :#: Go 2 :#: Go 1 :#: Turn :#: Go 4)
= Nil :#: Go 6 :#: Turn :#: Go 4
```

Credit may be given for indicating how you have tested your function. *[16 marks]*